CDF/MEMO/STATISTICS/PUBLIC/7587

Bayesian limit software: multi-channel with correlated backgrounds and efficiencies

Joel Heinrich—University of Pennsylvania

April 15, 2005

1 Introduction

Reference [1] described a Bayesian method for deriving limits from a single-channel Poisson counting experiment, where the acceptance and background are treated as nuisance parameters. The software associated with that approach is described in [2].

This note presents an approach to calculating Bayesian limits in a more general case, with multiple channels whose acceptance and background priors are modeled via Monte Carlo. This strategy permits correlations between the various acceptance and background priors (e.g. between channels).

The N different channels could refer to histogram bins, or different decay modes, or number of jets observed, or data sets with different triggers, etc. Correlations between the nuisance parameters might arise, for example, due to a common luminosity uncertainty among channels, or structure function uncertainties propagating into both signal and background, or any source of systematic uncertainty associated with more than one of the nuisance parameters.

The C code that implements this approach is available [3], and is also described in this note.

2 The Problem

Given N channels, and n_k observed events in the kth channel, $k = 1, 2, \ldots, N$, the Poisson probability of obtaining the observed result is

$$\prod_{k=1}^{N} \frac{e^{-(s\epsilon_k + b_k)}(s\epsilon_k + b_k)^{n_k}}{n_k!}$$

where s represents the parameter of interest (the cross section) and ϵ_k and b_k are the acceptance and expected background for the kth channel, respectively. (An "acceptance" parameter ϵ_k is typically a product of detector efficiency, branching fraction, and luminosity, and has units of inverse cross section.) All the ϵ_k and b_k have uncertainties and are considered "nuisance parameters". In the Bayesian approach, they are assigned priors, which may be correlated. So we write the joint nuisance prior as

$$\pi(\epsilon_1, b_1, \epsilon_2, b_2, \ldots, \epsilon_N, b_N) d\epsilon_1 db_1 d\epsilon_2 db_2 \cdots d\epsilon_N db_N$$

and the marginalized posterior for s is proportional to

$$\pi(s) \underbrace{\iint\cdots\int}_{(2N)} \pi(\epsilon_1, b_1, \epsilon_2, b_2, \ldots, \epsilon_N, b_N) \left[\prod_{k=1}^{N} \frac{e^{-(s\epsilon_k + b_k)} (s\epsilon_k + b_k)^{n_k}}{n_k!} \right] d\epsilon_1 db_1 d\epsilon_2 db_2 \cdots d\epsilon_N db_N$$

where $2N$ marginalization integrals are performed, and $\pi(s)$, the prior for s, is assumed for now to be independent of the joint nuisance prior.

3 Finite Bayesian Prior-Ensemble Approximation

Instead of doing the $2N$ marginalization integrals analytically, here we use Monte Carlo integration. We use the joint nuisance prior to generate (including all correlations) M random $(\epsilon_1, b_1, \epsilon_2, b_2, \ldots, \epsilon_N, b_N)$ vectors, which we save up in a big array (or ensemble)

$$\begin{array}{ccccccc}
\epsilon_{11} & b_{11} & \epsilon_{21} & b_{21} & \ldots & \epsilon_{N1} & b_{N1} \\
\epsilon_{12} & b_{12} & \epsilon_{22} & b_{22} & \ldots & \epsilon_{N2} & b_{N2} \\
\epsilon_{13} & b_{13} & \epsilon_{23} & b_{23} & \ldots & \epsilon_{N3} & b_{N3} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\epsilon_{1M} & b_{1M} & \epsilon_{2M} & b_{2M} & \ldots & \epsilon_{NM} & b_{NM}
\end{array}$$

where each line represents a single random vector of $2N$ nuisance parameters, and a second index for the line number has been inserted. Having stored such an array in computer memory, say filled with $2NM$ numbers of (C or C++) type `float`, this array, a finite Bayesian prior-ensemble, replaces the joint nuisance prior for all operations, in our approximation.

Given this ensemble, our marginalized posterior for s then becomes

$$p(s) = \frac{1}{\mathcal{N}} \frac{\pi(s)}{M} \sum_{i=1}^{M} \left[\prod_{k=1}^{N} \frac{e^{-(s\epsilon_{ki} + b_{ki})} (s\epsilon_{ki} + b_{ki})^{n_k}}{n_k!} \right]$$

where the normalization constant $\mathcal{N}$ is given by

$$\mathcal{N} = \int_0^\infty \frac{\pi(s)}{M} \sum_{i=1}^{M} \left[\prod_{k=1}^{N} \frac{e^{-(s\epsilon_{ki} + b_{ki})} (s\epsilon_{ki} + b_{ki})^{n_k}}{n_k!} \right] ds$$

4 Integrating the Marginalized Posterior

We specify an improper flat prior for s: $\pi(s) = 1$ for $s \geq 0$ and $\pi(s) = 0$ for $s < 0$. Then to calculate limits, we need to be able to evaluate the integral

$$\mathcal{I}(s_0) = \int_{s_0}^{\infty} \frac{1}{M} \sum_{i=1}^{M} \left[\prod_{k=1}^{N} \frac{e^{-(s\epsilon_{ki}+b_{ki})}(s\epsilon_{ki}+b_{ki})^{n_k}}{n_k!} \right] ds$$

which we rewrite as

$$\mathcal{I}(s_0) = \frac{1}{M} \sum_{i=1}^{M} \left[\int_{s_0}^{\infty} \prod_{k=1}^{N} \frac{e^{-(s\epsilon_{ki}+b_{ki})}(s\epsilon_{ki}+b_{ki})^{n_k}}{n_k!} ds \right]$$

We define

$$\epsilon'_i \equiv \sum_{k=1}^{N} \epsilon_{ki} \qquad\qquad b'_i \equiv \sum_{k=1}^{N} b_{ki}$$

which are the total acceptance and the total background (summed over the N channels) for the ith line of the ensemble. Then we have

$$\mathcal{I}(s_0) = \frac{1}{M} \sum_{i=1}^{M} \left[\int_{s_0}^{\infty} e^{-(s\epsilon'_i+b'_i)} \prod_{k=1}^{N} \frac{(s\epsilon_{ki}+b_{ki})^{n_k}}{n_k!} ds \right]$$

For each of the M integrals, we perform the substitution $x = (s - s_0)\epsilon'_i$, yielding

$$\mathcal{I}(s_0) = \frac{1}{M} \sum_{i=1}^{M} \left[\frac{e^{-(s_0\epsilon'_i+b'_i)}}{\epsilon'_i \prod_{k=1}^{N} n_k!} \int_0^{\infty} e^{-x} \prod_{k=1}^{N} \left(x\frac{\epsilon_{ki}}{\epsilon'_i} + s_0\epsilon_{ki} + b_{ki} \right)^{n_k} dx \right]$$

The product $\prod_{k=1}^{N} \left(x\frac{\epsilon_{ki}}{\epsilon'_i} + s_0\epsilon_{ki} + b_{ki} \right)^{n_k}$ appearing within the integral is just a polynomial in x of degree $n = \sum_{k=1}^{N} n_k$, so we need a method of evaluating the integral $\int_0^{\infty} e^{-x} f(x) dx$, where $f(x)$ is a polynomial of degree n. We will use Gauss-Laguerre quadrature[4], in which

$$\int_0^{\infty} e^{-x} f(x) dx = \sum_{j=1}^{J} w_j f(x_j) \qquad\qquad J = \lfloor n/2 \rfloor + 1$$

is exact for polynomials $f(x)$ of degree n or less. The "abscissas" x_j and the "weights" w_j are easily computed, x_j being the jth root of $L_J^0(x)$, the Laguerre polynomial of order J, for example. (Note that x_j and w_j depend on J.)

$$\mathcal{I}(s_0) = \frac{1}{M} \left[\sum_{i=1}^{M} \frac{e^{-(s_0\epsilon'_i+b'_i)}}{\epsilon'_i \prod_{k=1}^{N} n_k!} \left\{ \sum_{j=0}^{J} w_j \prod_{k=1}^{N} \left(x_j\frac{\epsilon_{ki}}{\epsilon'_i} + s_0\epsilon_{ki} + b_{ki} \right)^{n_k} \right\} \right]$$

Even though Gauss-Laguerre quadrature is usually classified as a numerical integration method, we are really integrating with respect to s analytically, since we are in precisely that case where Gauss-Laguerre quadrature is exact. This quadrature method is very efficient for small numbers of observed events. For example, if a total

of $n = 15$ events (spread out over N channels) are observed, only $J = 8$ evaluations of the function are necessary. For large n, we can truncate the sum over j at the point where next term is negligible, since the w_j become extremely small at large j.

Our upper limit $s_{\rm u}$ at credibility level β is then computed by solving

$$\mathcal{I}(s_{\rm u}) = (1-\beta)\mathcal{I}(0)$$

numerically using Newton's method. In some cases, it may be necessary to use a truncated flat prior for s: $\pi(s) = 1$ for $0 \le s \le s_{\rm max}$ and $\pi(s) = 0$ for $s < 0$ or $s > s_{\rm max}$. The upper limit is then obtained by solving

$$\mathcal{I}(s_{\rm u}) - \mathcal{I}(s_{\rm max}) = (1-\beta)[\mathcal{I}(0) - \mathcal{I}(s_{\rm max})]$$

for $s_{\rm u}$.

5 An Example

We illustrate the procedure with the following example. We specify 4 channels, with the prior for ϵ, the prior for b, and the number of observed events as given in this table:

chan	ϵ prior	b prior	events
1	1.0 ± 0.2	4.00 ± 2.00	6
2	1.0 ± 0.1	1.60 ± 0.40	1
3	1.0 ± 0.01	0.25 ± 0.05	1
4	1.0 ± 0.2	5.00 ± 1.00	8

We further specify all eight priors as gamma distributions, with the central value and uncertainty equal to the mean and standard deviation of the gamma distributions. They are all taken to be independent in our example, except for the background priors for channels 3 and 4, which are taken to be linearly dependent: we specify that the expected background in channel 4 is exactly 20 times the expected background in channel 3 (although neither is known precisely).

We then generate a prior ensemble, which in our example looks something like this:

ϵ_1	b_1	ϵ_2	b_2	ϵ_3	b_3	ϵ_4	b_4
1.1416	4.7548	1.0372	1.4205	1.0041	0.2347	1.0869	4.6935
0.8820	4.1672	1.1450	1.0760	0.9923	0.2351	1.1707	4.7025
1.1442	1.1252	1.0081	2.0481	0.9977	0.3270	1.1986	6.5394
1.2089	4.6549	0.9685	1.9612	0.9994	0.2266	1.4397	4.5329
1.0559	2.1078	0.9651	1.7600	0.9906	0.2176	0.8316	4.3526
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Taking the ensemble size to be $M = 150000$, we obtain the associated 95% upper limit $s_{\rm u}$. For illustration, we repeat this five times (and also five times with $M = 1500$) generating a new ensemble for each repetition, obtaining:

$M = 150000$	$M = 1500$
2.8738 ± 0.0011	2.8799 ± 0.0106
2.8764 ± 0.0011	2.8714 ± 0.0107
2.8746 ± 0.0011	2.8859 ± 0.0109
2.8765 ± 0.0011	2.8791 ± 0.0104
2.8765 ± 0.0011	2.8885 ± 0.0110

The table also illustrates an additional feature of the code that calculates the upper limit; it estimates the statistical uncertainty (RMS) of the limit due to the finite Monte Carlo sample size. Attempts to try to fold back the uncertainty of the numerical value of the upper limit into the upper limit itself are considered nonstandard and unnecessary; if more significant digits are needed, a larger value of M can be used. A more time and memory consuming run with an ensemble size of 5×10^6 yields a 95% upper limit of 2.8764 ± 0.0002. The particular case that we have chosen, with rather large uncertainties on ϵ and b for channel 1, requires a relatively large M to establish the upper limit to high accuracy; more typical cases with smaller prior uncertainties will permit M to be smaller as well.

It is also important to look at the shape of the posterior p.d.f. for s. Code is available to calculate the value of the posterior at a given s. In Fig. 1 we plot the posterior twice, calculated using two independently generated prior ensembles of size $M = 100$. The small sample size was chosen to make the Monte Carlo fluctuations visible; with more reasonable values of M, the curves lie on top of one another.

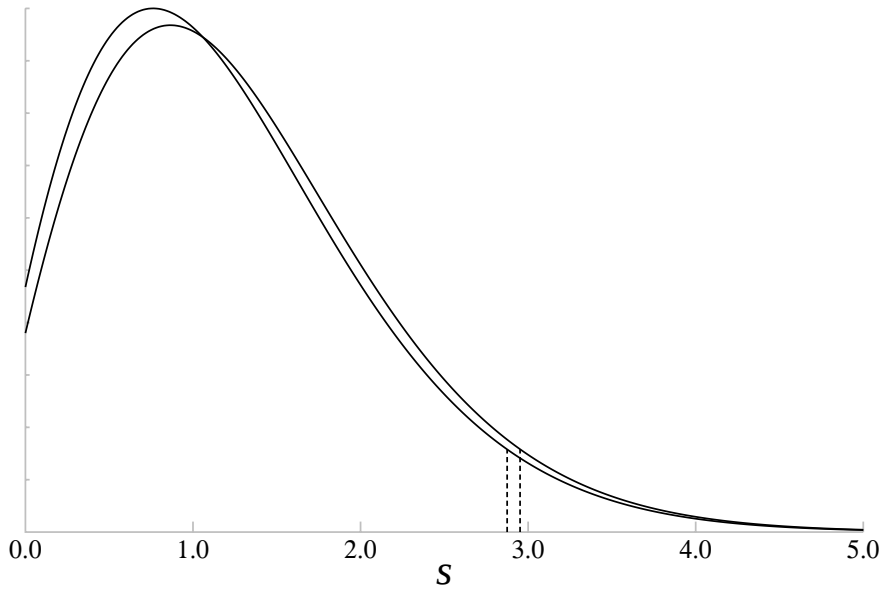

Figure 1: Posterior $p(s)$ and 95% upper limit from two independent prior ensembles of size $M = 100$. (Small M chosen to exaggerate fluctuation due to Monte Carlo statistics.)

6 An Alternative Cross Section Prior

A "correlated prior" method is suggested for the multiple channel case in [5]. To implement that scheme here, we replace the joint nuisance prior in the expressions of the previous section by

$$\epsilon' \pi(\epsilon_1, b_1, \epsilon_2, b_2, \ldots, \epsilon_N, b_N) d\epsilon_1 db_1 d\epsilon_2 db_2 \cdots d\epsilon_N db_N$$

That is, we insert an additional factor of ϵ' (the total acceptance). We will keep the prior ensemble as previously defined, i.e. generated from $\pi(\epsilon_1, b_1, \epsilon_2, b_2, \ldots, \epsilon_N, b_N)$ as before, and take the additional factor of ϵ' as an extra numerical weight. This allows us, for example, to compute limits using the flat (uncorrelated) cross section prior and the corresponding correlated prior without regenerating the prior ensemble.

Our expression for the posterior p.d.f. then becomes

$$p_{\rm c}(s) = \frac{1}{\mathcal{I}_{\rm c}(0) M} \sum_{i=1}^{M} \left[\epsilon'_i \prod_{k=1}^{N} \frac{e^{-(s\epsilon_{ki}+b_{ki})}(s\epsilon_{ki}+b_{ki})^{n_k}}{n_k!} \right]$$

and its integral is computed using

$$\mathcal{I}_{\rm c}(s_0) = \frac{1}{M} \left[\sum_{i=1}^{M} \frac{e^{-(s_0\epsilon'_i+b'_i)}}{\prod_{k=1}^{N} n_k!} \left\{ \sum_{j=0}^{J} w_j \prod_{k=1}^{N} \left(x_j \frac{\epsilon_{ki}}{\epsilon'_i} + s_0\epsilon_{ki} + b_{ki} \right)^{n_k} \right\} \right]$$

(note the cancellation of an ϵ'_i in the denominator), where the subscript label "c" has been added to denote the alternative "correlated" prior.

In addition to the benefits described in [5], the correlated prior will normally yield a smaller numerical uncertainty on the value of a limit (due to Monte Carlo fluctuations in the prior ensemble) than will the corresponding flat cross section prior. This happens because the flat uncorrelated prior gives the ith line of the prior ensemble an effective "weight" in the posterior of $\sim 1/\epsilon'_i$, while this weight is ~ 1 when using the correlated prior. This effect becomes most noticeable when the uncertainties on the acceptance parameters are all relatively large, so that ϵ'_i occasionally fluctuates close to zero.

7 The Software

The basic philosophy of the software is that the user is responsible for writing the code to construct the prior ensemble in the expected format. All memory allocation and deallocation is the responsibility of the user. The provided code will then handle computing the posterior, integrating the posterior, and calculating limits. Although the code is written in C, care was taken to ensure that the code will also compile and run as C++. The header file `genlimit.h` declares all the associated functions. The code is available from [3].

7.1 the prior ensemble

The prior ensemble is stored as an array of `EB`, where `EB` is defined as:

```
typedef struct {
  float e,b;
}EB;
```

The user is responsible for allocating sufficient space to store the ensemble. If we define `nchan` as the number of channels N, and `nens` as the number of lines M in the ensemble, then

```
EB* ens = (EB*)malloc(nchan*nens*sizeof(EB));
```

in C, or

```
EB* ens = new EB[nchan*nens];
```

in C++ would create a pointer `ens` to a newly allocated block of memory with the proper size to hold the ensemble. Schematically, filling the ensemble is illustrated by this user code fragment

```
k=0;
for(i=0;i<nens;++i) {
   for(j=0;j<nchan;++j) {
      ens[k].e = acceptance(j);
      ens[k].b = background(j);
      ++k;
   }
}
```

where `acceptance(j)` and `background(j)` (which would be written by the user) are assumed to provide random deviates appropriate for the `j`th channel (numbered `0` to `nchan-1` in the code). For simplicity, we omit any complications due to correlations in this fragment. The file `example.c` provides a more general example with a correlation between two channels.

7.2 the posterior p.d.f.

The value of the posterior p.d.f. at `s` is returned by

```
double cspdf(double s,double norm,int nchan,
             int nens,const int nobs[],const EB* ens,PRIOR prior);
```

where the user must supply the array `nobs[]` which holds the number of observed events in each channel, and `norm`, which is the value of $\mathcal{I}(0)$ (calculated by the code described in the next section). The last argument is a switch allowing the user to specify whether to use the basic flat cross section prior or the correlated version. The two allowed values are `flat` and `corr`, defined in `genlimit.h` via the enumeration

```
typedef enum {
  flat=10,
  corr=20
} PRIOR;
```

7.3 integrating the posterior

The basic routine for evaluating the function $\mathcal{I}(s_0)$ is

```
double csint(double s0,int nchan,int nens,const int nobs[],const EB* ens,
             int* ngl,double xgl[],double lwgl[],
             PRIOR prior,double* uncertainty);
```

The memory pointed to by the arguments `ngl`, `xgl`, `lwgl`, and `uncertainty` may be written to by `csint`, which requires some explanation. Executing the code

```
int ngl = 0;
double xgl[500], lwgl[500];
double uncer = 0;
double norm = csint(0,nchan,nens,nobs,ens,&ngl,xgl,lwgl,flat,&uncer);
```

will result in J abscissas x_j and J logarithms of weights $\ln(w_j)$ associated with the Gauss-Laguerre quadrature scheme being stored in `xgl` and `lwgl` respectively, and the value of J being stored in `ngl`. In this example, the user is responsible for ensuring that `xgl` and `lwgl` are dimensioned large enough so that `ngl <= 500` after the call. An estimate of the uncertainty of `norm` ($\mathcal{I}(0)$ in this example) due to Monte Carlo statistical fluctuation is stored in `uncer` (the last argument may also be a null pointer, in which case no uncertainty is returned).

On succeeding calls, the code will see that `ngl > 0` and will not recalculate the abscissas and log-weights; this is important since their calculation is time consuming. **Should n, the total number of events observed, change, the user should set `ngl` back to zero so that `xgl` and `lwgl` will be recalculated. Failure to set `ngl=0` when n changes will merely cause unnecessary extra computation when n is decreased, but may lead to inaccurate results when n is increased.**

More specialized functions are also provided:

```
void csint2(double s1,double s2,
    int nchan,int nens,const int nobs[],const EB* ens,
    int* ngl,double xgl[],double lwgl[],PRIOR prior,
    double* int1,double* int2,
    double* v11,double* v12,double* v22);
```

stores $\mathcal{I}(s_1)$ in `*int1`, $\mathcal{I}(s_2)$ in `*int2`, and their covariance matrix (for MC fluctuations) is stored in `*v11`, `*v12`, and `*v22`. The function

```
void csint2cut(double s1,double s2,double shi,
       int nchan,int nens,const int nobs[],const EB* ens,
       int* ngl,double xgl[],double lwgl[],PRIOR prior,
       double* int1,double* int2,
       double* v11,double* v12,double* v22);
```

stores $\mathcal{I}(s_1)-\mathcal{I}(s_{\rm hi})$ in `*int1`, $\mathcal{I}(s_2)-\mathcal{I}(s_{\rm hi})$ in `*int2`, and their covariance matrix (for MC fluctuations) in `*v11`, `*v12`, and `*v22`.

7.4 limit calculation

The function

```
double cslimit(double beta,int nchan,int nens,const int nobs[],const EB* ens,
               int* ngl,double xgl[],double lwgl[],
               PRIOR prior,double* uncertainty);
```

returns the upper limit $s_{\rm u}$ at credibility level β, obtained by solving $\mathcal{I}(s_{\rm u}) = (1-\beta)\mathcal{I}(0)$ numerically. In the corresponding case with a flat prior for s cutoff at $s_{\rm max}$, the upper limit is returned by

```
double cscutlimit(double beta,double smax,
                  int nchan,int nens,const int nobs[],const EB* ens,
                  int* ngl,double xgl[],double lwgl[],
                  PRIOR prior,double* uncertainty);
```

In each case, the uncertainty in the limit due to MC fluctuation is also returned (unless the last argument is a null pointer). **See the previous subsection for instructions concerning `ngl`.**

7.5 auxiliary functions

The function

```
void gausslaguerre(double x[],double lw[],int n,double alpha);
```

does the actual work of calculating the abscissas and log-weights associated with Gauss-Laguerre quadrature. The function

```
double galim(double beta,int nchan,int nens,const int nobs[],const EB* ens);
```

returns a crude Gaussian approximation to the upper limit, which is used as a starting point by the `cslimit` routines.

7.6 the example code

The code that calculates the limits for the example of section 5 is provided in the file `example.c`. There are four functions associated with random number generation involved:

```
double rlx();
void rlxrandomize();
double gamma_gen(double a,double (*ru)());
double gauss_gen(double (*ru)());
```

The functions `rlx` and `rlxrandomize` are interfaces to the fortran CERNLIB `ranlux` uniform generator, and `gamma_gen` and `gauss_gen` provide random deviates from the gamma and Gaussian distributions, respectively. As these functions are only used by `example.c` for constructing its prior ensemble, and are not considered logically part of this limit setting software package, their declarations are not in `genlimit.h`.

References

[1] Joel Heinrich et al., "Interval estimation in the presence of nuisance parameters. 1. Bayesian approach.", CDF Internal Note 7117, (2004), www-cdf.fnal.gov/publications/cdf7117_bayesianlimit.pdf.

[2] Joel Heinrich, "User Guide to Bayesian-Limit Software Package", CDF Internal Note 7232, (2004), www-cdf.fnal.gov/publications/cdf7232_blimitguide.pdf; www-cdf.fnal.gov/physics/statistics/statistics_software.html.

[3] www-cdf.fnal.gov/physics/statistics/statistics_software.html

[4] William H. Press, et al., "Numerical Recipes", 2nd edition, (Cambridge University Press, Cambridge, 1992), §4.5, lib-www.lanl.gov/numerical/bookcpdf/c4-5.pdf; Eric W. Weisstein, "Laguerre-Gauss Quadrature", from MathWorld—A Wolfram Web Resource, mathworld.wolfram.com/Laguerre-GaussQuadrature.html.

[5] Luc Demortier, "A Fully Bayesian Computation of Upper Limits for Poisson Processes" CDF note 5928, October 2004. www-cdf.fnal.gov/publications/cdf5928_Bayesian_upperlimits.ps